APPENDIX A

```
**************************************************
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <assert.h>
#include <bitvector.h>
#include <list.h>

struct Procedure : public Object {
 char *code ;
 char *desc ;
 double fee ;
 int isPanel ;   // indicates whether procedure is a panel
 BitVector panelProcs ; // if is panel, this contains the panel codes
} ;

struct TreeNode : public Object {
 BitVector procedures ;  // list of procedures for this branch
 List branches ;
} ;

int numProcedures = 0 ;
Procedure *procedures = NULL ; // array of procedures
TreeNode rootNode ;

int findCode(char *code)
{
 for(int i=0;i<numProcedures;i++)
  if( strcmpi(procedures[i].code,code)==0) return i ;
 return -1 ;
}

void trimString(char *str)
{
 char *p = str ;
 while(isspace(*p)) p++ ;
 int len = strlen(p) ;
 memcpy(str,p,len) ;
 str[len] = 0 ;
```

```
len = strlen(str) ;
if( len ) {
 p = str+(strlen(str)-1) ;
 while(p>=str &&isspace(*p)) *p-- = 0 ;
}
}

void loadPanelDefinitions(char *filename)
{
 FILE *fp = fopen(filename,"r") ;
 assert(fp) ;

 char buf[1024] ;

 // load the panel definitions
 while(fgets(buf,sizeof(buf),fp)) {
  trimString(buf) ;

  // get code and procedure
  char *p = strtok(buf,", ") ;
  int id = findCode(p) ;
  if( id >= 0 ) {
   Procedure *panel = &procedures[id] ;
   panel->isPanel = TRUE ;
   while(p=strtok(NULL,", ")) {
    int pid = findCode(p) ;
    if( pid >= 0 )
     panel->panelProcs.set(pid) ;
   }
  }
 }

 fclose(fp) ;
}

void buildProcedureList(char *filename)
{
 FILE *fp = fopen(filename,"r") ;
 assert(fp) ;

 char buf[1024] ;
```

```
// get the first line containing the number of procedures
fgets(buf,sizeof(buf),fp) ;
numProcedures = atoi(buf) ;
assert(numProcedures>0&&numProcedures<20000) ; // the 20000 cap is
arbitrary just for sanity's sake

// allocate the array of codes
procedures = new Procedure[numProcedures] ;
assert(procedures) ;

// load the procedures
int i = 0 ;
while(fgets(buf,sizeof(buf),fp)) {
 trimString(buf) ;

 // get code
 char *p = strtok(buf,", ") ;
 procedures[i].code = strdup(p) ;
 procedures[i].isPanel = FALSE ;

 // get fee
 p = strtok(NULL,", ") ;
 procedures[i].fee = atof(p) ;

 // get desc
 p = strtok(NULL,"") ;
 if( p ) {
  while(*p&&(isspace(*p)||*p==',')) p++ ;
  procedures[i].desc = strdup(p) ;
 }else
  procedures[i].desc = strdup("") ;

//printf("%s,%f,%s\n",procedures[i].code,procedures[i].fee,procedures[i].des
c) ;

 i++ ;
 }

 fclose(fp) ;
}

BitVector *loadOrders(char *filename)
```

```
{
BitVector *orders = new BitVector ;
assert(orders) ;

FILE *fp = fopen(filename,"r") ;
assert(fp) ;

char buf[1024] ;

// load the panel definitions
while(fgets(buf,sizeof(buf),fp)) {
.trimString(buf) ;

 // get code and procedure
 char *p = strtok(buf,", ") ;
 int id = findCode(p) ;
 assert(id>=0) ;  // Invalid procedure code in ordered procedures
 if( id >= 0 ) orders->set(id) ;
}

fclose(fp) ;
return orders ;
}

void unbundlePanels(BitVector *procList)
{
// this routine makes multiple passes through orders list in case
// there is an occurance of a panel within a panel
int changed ;
do{
 changed = FALSE ;
 for(int i=procList->start();i<=procList->end();i++) {
  if( procList->get(i) &&
   procedures[i].isPanel ) {
   procList->clear(i) ;
   procList->or(&procedures[i].panelProcs) ;
   changed = TRUE ;
  }
 }
} while(changed) ;
}
```

```
void dumpProcedureList(BitVector *procList)
{
 int numPanels = 0 ;
 int numProcs = 0 ;
 double totalFee = 0.0 ;

 // dump ordered procedures
 for(int i=procList->start();i<=procList->end();i++) {
  if( procList->get(i) ) {
   numProcs++ ;
   totalFee += procedures[i].fee ;
   if( procedures[i].isPanel ) numPanels++ ;
   printf("%s - %s %s\n",
    procedures[i].code,procedures[i].desc,(char
*)((procedures[i].isPanel)?"(PANEL)":"")) ;
  }
 }
 printf("\n") ;
 printf("Total panels: %d\n", numPanels) ;
 printf("Total procedures: %d\n", numProcs) ;
 printf("Total Fee for Listed Procedures: $%-5.2f\n", totalFee) ;
}

int panelProceduresMatch(Procedure *panelProc,BitVector *procList)
{
 // first check to see if panel proc is special chemistry procedure
 BitVector bv ;
 bv.or(&panelProc->panelProcs) ;
 bv.and(procList) ;

 //printf("****Dumping merged procedure list****\n\n") ;
 //printf("Panel Procs\n") ;
 //dumpProcedureList(&panelProc->panelProcs) ;
 //printf("procList\n") ;
 //dumpProcedureList(procList) ;
 //printf("Merged\n") ;
 //dumpProcedureList(&bv) ;
 //printf("**** END OF DUMP ****\n\n") ;
 for(int i=bv.start();i<=bv.end();i++) {
  if( bv.get(i) ) {
   printf("  Matched %s on procedure
%s\n",panelProc->code,procedures[i].code) ;
```

```
  return TRUE ;
 }
}
return FALSE ;
}

void dumpTree(TreeNode *node,int lvl)
{
TreeNode *pNode = (TreeNode *)node->branches.First() ;
int cnt = 0 ;
while(pNode) {
 printf("%*.*s%d:%d:",lvl,lvl,"
",lvl,cnt) ;
 for(int i=pNode->procedures.start();i<=pNode->procedures.end();i++) {
  if( pNode->procedures.get(i) )
   printf("%s,",procedures[i].code) ;
  }
 printf("*\n") ;
 dumpTree(pNode,lvl+1) ;
 cnt ++ ;
 pNode = (TreeNode *)pNode->next ;
}
}

void buildTree(TreeNode *node,BitVector *procList)
{
TreeNode *pNode = new TreeNode ;
assert(pNode) ;
pNode->procedures.or(procList) ;
node->branches.Insert(pNode) ;
for(int i=0;i<numProcedures;i++) {
 if( procedures[i].isPanel &&
  panelProceduresMatch(&procedures[i],procList) ) {
//   printf("Matched %s\n",procedures[i].desc) ;
  BitVector panelProcList(numProcedures,TRUE) ;
  panelProcList.clearAll() ;
  panelProcList.or(&procedures[i].panelProcs) ;
  panelProcList.and(procList) ;
  pNode = new TreeNode ;
  assert(pNode) ;
  pNode->procedures.set(i) ; // set the panel ID
  node->branches.Insert(pNode) ;
```

```
  BitVector remainingProcList(numProcedures,TRUE) ;
  remainingProcList.or(procList) ;
  for(int j=panelProcList.start();j<=panelProcList.end();j++)
   if( panelProcList.get(j) ) remainingProcList.clear(j) ;
  if( !remainingProcList.isNull() )
   buildTree(pNode,&remainingProcList) ;
 }
}
}

double calcFee(BitVector *procList)
{
 double totalFee = 0.0 ;

 // dump ordered procedures
 for(int i=procList->start();i<=procList->end();i++)
  if( procList->get(i) )
   totalFee += procedures[i].fee ;
 return totalFee ;
}

double getMinimumFee(TreeNode *node,BitVector *minProcs)
{
 TreeNode *pNode = (TreeNode *)node->branches.First() ;
 TreeNode *minNode = NULL ;
 double fee = 0.0 ;
 BitVector minVector(numProcedures,TRUE) ;

 while(pNode) {
  double val = calcFee(&pNode->procedures) ;
  BitVector bv(numProcedures,TRUE) ;
  bv.clearAll() ;
  bv.or(&pNode->procedures) ;
  val = val + getMinimumFee(pNode,&bv) ;
  if( !minNode || val < fee ) {
   minNode = pNode ;
   fee = val ;
   minVector.clearAll() ;
   minVector.or(&bv) ;
  }
  pNode = (TreeNode *)pNode->next ;
 }
```

```
minProcs->or(&minVector) ;
return fee ;
}

int main(int argc, char *argv[])
{
printf("Dynamedix Medicare Billing Optimization Patent Algorithm\nUsing
Dallas Locale and Carrier Code For Fee Structure") ;

buildProcedureList("codes.txt") ;
loadPanelDefinitions("panels.txt") ;

// load the orders, build the tree and report the results
BitVector *orders = loadOrders("orders.txt")  ;

// Unbundle and dump
printf("\nOriginal Order\n\n") ;
dumpProcedureList(orders) ;
unbundlePanels(orders) ;
printf("\nOriginal Order Unbundled with Duplicates Removed\n\n") ;
dumpProcedureList(orders) ;

printf("\n\nBuilding Tree Structure\n") ;
TreeNode root ;
buildTree(&root,orders) ;
dumpTree(&root,1) ;

BitVector optOrder(numProcedures,TRUE) ;
double fee = getMinimumFee(&root,&optOrder) ;
printf("\n\nOptimum Order\n") ;
dumpProcedureList(&optOrder) ;

return 0 ;
}
```